Security regression visualization in CI/CD using trivy and GitHub actions*

Yurii Zhuravchak[1,*,†], Danyil Zhuravchak[1,†], Anastasiia Zhuravchak[1,†] and Ivan Opirskyy[1,†]

[1] Lviv Polytechnic National University, Information Security Department, 12 Stepan Bandera str., 79000 Lviv, Ukraine

Abstract

This paper presents a lightweight and reproducible method for integrating container security scanning into CI/CD pipelines using fully open-source components. While many modern DevSecOps workflows incorporate vulnerability scanners such as Trivy, their output is typically used in a point-in-time manne —alerts are generated, acted upon, and forgotten. In contrast, this work introduces the concept of security regression tracking: a simple yet effective mechanism to persist, visualize, and analyze vulnerability trends across multiple builds over time. Our approach logs the number of vulnerabilities detected at each severity level (Critical, High, Medium, Low) per build, stores them in a time-series CSV file, and renders them using a Chart.js-based dashboard hosted on GitHub Pages. This enables development teams to assess whether their security posture is improving or regressing as images evolve, dependencies change, and patches are applied. To demonstrate the viability of the system, we conduct a structured experiment using a deliberately vulnerable container image. We simulate five typical development stages, including partial fixes, regressions, and full remediations. The dashboard clearly illustrates the impact of these changes, offering visual feedback that can guide technical decisions and risk prioritization. This solution is infrastructure-free, transparent, and extensible, making it well suited for small to medium-sized teams, open-source projects, and educational use. By shifting vulnerability tracking from reactive to historical and strategic, it empowers DevSecOps teams with actionable temporal insight.

Keywords

DevSecOps, CI/CD, Trivy, vulnerability scanning, GitHub Actions, dashboard, security regression, open source

1. Introduction

The acceleration of software development through Continuous Integration and Continuous Deployment (CI/CD) pipelines has profoundly reshaped the way modern applications are built, tested, and delivered. These practices enable rapid iteration, frequent releases, and faster innovation—but they also introduce new security risks [1–4]. In environments where container images are rebuilt and deployed multiple times a day, even a small vulnerability can rapidly propagate into production unless adequate controls are in place [5, 6].

In response to these concerns, the DevSecOps paradigm promotes the integration of security measures directly into the development pipeline [7]. Rather than applying security as a final checkpoint, DevSecOps encourages embedding tools for static and dynamic analysis early and continuously. Open-source scanners such as Trivy [8], Grype [9], and Semgrep make it possible to detect vulnerabilities in codebases, dependencies, and container images as part of the automated build process [10].

However, despite the availability of these tools, most scanning implementations operate in a point-in-time fashion: a vulnerability is detected, an alert is issued, and developers either fix it or suppress it. There is rarely any mechanism to retain and analyze scan data across builds, nor to assess whether the overall security posture is improving or regressing over time. This temporal

* CSDP'2025: Cyber Security and Data Protection, July 31, 2025, Lviv, Ukraine

* Corresponding author.

† These authors contributed equally.

yurii.zhuravchak.mitis.2023@lpnu.ua (Y. Zhuravchak); danyil.y.zhuravchak@lpnu.ua (D. Zhuravchak); anastasiia.y.tolkachova@lpnu.ua (A. Zhuravchak); ivan.r.opirskyi@lpnu.ua (I. Opirskyy)

0009-0003-2378-5365 (Y. Zhuravchak); 0000-0003-4989-0203 (D. Zhuravchak); 0000-0002-8196-7963 (A. Zhuravchak); 0000-0002-8461-8996 (I. Opirskyy)

blindness creates a missed opportunity for strategic decision-making, especially in long-running projects with frequent deployments and changing dependencies [11].

In real-world development, the security profile of an application evolves continuously. Base images are patched (or go unpatched), third-party libraries are updated, and insecure patterns may be introduced or removed. Without visibility into these changes, teams are forced to react to each issue in isolation, with no sense of whether their system is becoming more secure or accumulating technical security debt [12].

In this paper, we propose a lightweight, fully open-source approach for security regression tracking in containerized applications. Using Trivy by Aqua Security [8], we integrate vulnerability scanning directly into the CI/CD pipeline via GitHub Actions [13]. Each scan extracts severity-level vulnerability counts (Critical, High, Medium, Low), which are appended to a time-series CSV log. This data is then rendered using a web-based dashboard (Chart.js) that visualizes how vulnerability levels change across builds.

Our system is:

1. Self-contained and infrastructure-free (hosted entirely within the GitHub repository).
2. Reproducible, with every scan logged and versioned.
3. And extensible, allowing future integrations such as SBOM correlation, badge generation, or full CVE tracing.

To validate the approach, we conducted an experiment with a deliberately vulnerable container image. We introduced five progressive modifications, mimicking common development scenarios—partial fixes, regressions, dependency updates—and tracked the resulting scan output over time. This demonstrated the system's ability to capture both improvements and degradations in a project's security posture.

The primary goal of this research is to develop and evaluate a practical and transparent method to visualize the evolution of vulnerabilities in container images over time—something that is rarely supported out of the box in existing security tools. We aim to bridge the gap between DevSecOps automation and historical awareness, empowering development teams to track not just if they are secure, but how that security is evolving with each build.

2. Related work

The integration of security scanning within CI/CD pipelines has become a cornerstone of modern DevSecOps practices [14]. Numerous tools—including Trivy, Grype, and Snyk—offer automated vulnerability scanning at build time, enabling teams to shift security left and detect issues early in the software development lifecycle (SDLC). However, while static analysis and container scanning are now widely adopted, relatively few approaches provide mechanisms for tracking how security posture evolves over time.

A growing body of research has addressed vulnerability detection in container ecosystems. These studies typically focus on quantifying vulnerability prevalence, comparing scanner effectiveness, and assessing the impact of base-image choices [15].

Wist, Helsem, and Gligoroski [16] analyzed more than 2,500 Docker Hub images and discovered a high density of vulnerabilities in publicly available containers. Their findings showed that even official images are not immune to risk, though they generally fared better than community-maintained variants. Importantly, they also highlighted a lack of correlation between image popularity and security, which implies that trusted sources may still harbor critical flaws.

Haque and Babar [17], in their empirical study Well Begun is Half Done, explored base image inheritance chains across 64,579 real-world GitHub projects. They found that a small number of vulnerable base images led to wide vulnerability propagation downstream, emphasizing the cascading risk of unpatched root layers. Their analysis demonstrates the importance of selecting

and maintaining secure foundational images—yet, again, their focus was on static analysis, not temporal tracking.

Kaur et al. [18] extended this line of inquiry into scientific computing environments, evaluating container images used in disciplines such as neuroscience. Using four scanning tools including Trivy, they demonstrated that regular updates and minimal image design could reduce vulnerabilities by up to two-thirds. This reinforces the practice of image hygiene but also showed that scanner outputs vary significantly, introducing complexity when comparing risk across time or tools.

Another critical angle is explored by Zerouali et al. [19], who introduced the notion of technical lag—the difference between package versions in containers and their upstream releases. They studied over 7,300 Debian-based containers and concluded that even "latest" tags are often outdated, contributing to latent vulnerabilities and bug exposure. This illustrates that freshness alone is not a guarantee of security and that visibility into versioning gaps is essential.

Despite these contributions, most existing research provides a static snapshot of security status. Studies often benchmark a set of containers at one point in time without exploring how vulnerability profiles shift due to code changes, dependency updates, or changes in upstream images. As such, temporal awareness—how vulnerabilities increase or decrease across commits or builds—remains underexplored.

In addition to container-specific studies, recent research emphasizes the importance of integrating multi-layered security approaches within complex software ecosystems. Milevskyi et al. [20] propose a multi-contour security methodology for sociocyberphysical systems, highlighting the benefits of combining automated detection with layered protective controls. Similarly, Shevchuk et al. [21] demonstrate the design of secured authentication, authorization, and accounting services, which can be extended to CI/CD pipelines to enforce access policies and prevent misconfigurations that may introduce vulnerabilities. Lakhno et al. [22] further stress the role of decision-support systems in managing information protection, suggesting that automated guidance can improve security response times and reduce human error. Additionally, Vakhula et al. [23] explore cloud security challenges under dynamic orchestration, advocating for "security-as-code" approaches that can complement vulnerability tracking over time. Finally, Harasymchuk et al. [24, 25] show that structured information classification frameworks and predictive assessment with LLMs can aid in prioritizing remediation efforts, which aligns with the objective of monitoring how vulnerability profiles evolve across CI/CD builds.

Furthermore, few open-source projects address this gap in a reproducible, dashboard-based format that integrates directly with CI/CD. Available enterprise solutions (e.g., Snyk Monitor, GitHub Advanced Security) offer partial historical views, but they are often opaque, require paid plans, and do not easily integrate into custom DevSecOps workflows.

This paper proposes a solution to fill this void: a lightweight, automated, and fully open method for vulnerability trend monitoring in CI/CD. By leveraging Trivy for scanning, GitHub Actions for automation, and Chart.js [25] for visualization, we create a transparent and extensible framework that tracks vulnerability counts over time. This enables teams to assess not just whether vulnerabilities exist, but how their posture is evolving—bringing measurable, historical awareness into security automation.

3. Methodology

3.1. CI/CD Integration

We used GitHub Actions as the automation platform for our pipeline due to its accessibility, tight integration with version control, and support for public repositories. The pipeline is triggered either on code pushes to the main branch or on a scheduled daily basis.

The pipeline performs the following steps:

1. Docker image build: using a Dockerfile, we generate the container image to be tested.
2. Security scan: we run Trivy in JSON [26] output mode to scan the image for vulnerabilities in operating system packages and application dependencies.
3. Parsing step: a custom Python script extracts key statistics from Trivy's output (number of Critical, High, Medium, and Low vulnerabilities) and appends them as a new row in a CSV file [27].

3.2. Scan Result Archiving

To enable tracking over time, the pipeline stores results in a CSV file (history.csv), where each row corresponds to a specific scan. The format includes a UTC timestamp and severity-level counts:

```
timestamp,CRITICAL,HIGH,MEDIUM,LOW
2025-07-26 08:36:20,2,1,4,9
2025-07-26 08:38:18,0,1,4,22
2025-07-26 08:40:35,0,5,4,10
```

Figure 1: history.csv file

The Python script appends new scan data to the file and ensures that only the most recent seven entries are retained. This windowed history simplifies visualization while capturing trends in vulnerability evolution.

To persist this history across runs, the workflow restores the previous docs/data.csv file (committed to the repository) and copies it into the working directory before appending new results.

3.3. Scan Result Archiving

For intuitive tracking of risk changes, we developed a web-based dashboard using static HTML and the Chart.js JavaScript library [28]. The dashboard reads data.csv from the /docs/ folder via a client-side CSV parser and renders two visualizations:

- A line chart showing changes in vulnerability count over time.
- A point-only chart that emphasizes identical data points to reveal when values remain unchanged.

In addition, we compute and display a mathematical summary below the charts. This includes:

- The difference in vulnerability counts from the first to the last scan.
- Whether each category (e.g., Critical) has improved, worsened, or remained constant.
- The total change in overall vulnerability counts.

This summary gives immediate insight into the effectiveness of recent remediation efforts or regressions.

3.4. Experiment Design

To simulate real-world security posture evolution, we created a sequence of five Docker image [29] versions with controlled modifications:

Table 1
Docker image versions with controlled modifications

Version	Description	Expected Trend
v1	Baseline image with known vulnerable packages	High vulnerability count
v2	Some vulnerable packages removed	Slight improvement
v3	New risky dependencies added	Spike in vulnerabilities
v4	Updated base image and minimized packages	Strong improvement
v5	Clean Alpine-based image with minimal footprint	Very low vulnerability count

Each image is built and scanned in order, with results logged and visualized. This controlled scenario emulates typical development lifecycle iterations and demonstrates the system's ability to track both improvement and degradation.

4. Results

To evaluate the effectiveness of our approach, we applied the pipeline to a series of deliberately crafted Docker images (v1 to v5), each representing different stages of software hardening or regression. For each version, a Trivy scan was performed via GitHub Actions, and results were appended to a CSV log, forming the data foundation for trend visualization.

4.1. Vulnerability Trends over Time

The dashboard rendered the following timeline of scan results, displaying changes in vulnerability counts segmented by severity:

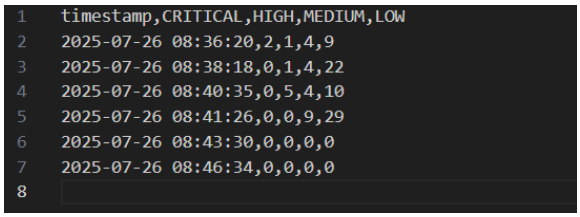

Figure 2: file with history results

- v1 shows a high baseline of vulnerabilities across all severities.
- v2 demonstrates slight improvements due to the removal of several risky packages.
- v3 introduces a spike, reflecting added vulnerable dependencies.
- v4 significantly reduces all counts by replacing the base image and optimizing installed packages.
- v5 results in near-clean output using Alpine and minimal footprint design.
- V6 is v5.

These changes are visualized via a line and point-based chart, clearly reflecting both security regressions (v3) and improvements (v4, v5).

4.2. Mathematical Summary

An automatic calculation in the dashboard summarizes the difference between the first and last scan:

```
Critical: 0 → 0 (→ no change, Δ = 0)
High    : 1 → 0 (↓ improved, Δ = -1)
Medium  : 4 → 0 (↓ improved, Δ = -4)
Low     : 22 → 0 (↓ improved, Δ = -22)

Total     : 27 → 0 (✅ Overall improved, Δ = -27)
```

Figure 3: Vulnerability Improvement Summary

This quantitative summary confirms the effectiveness of progressive hardening steps and validates the system's ability to track security posture changes in a meaningful and interpretable manner.

4.3. Dashboard Evaluation

The final dashboard provides:

- Clear visual representation of vulnerability fluctuations.
- Contextual feedback on security regression or improvement.
- Instant visibility into build history with minimal manual effort.

Despite its simplicity, this solution proved to be highly effective in practice. The use of GitHub-hosted infrastructure enabled easy sharing and reproduction of results, while the use of static CSV logging avoided the complexity of full-fledged databases or dashboards.

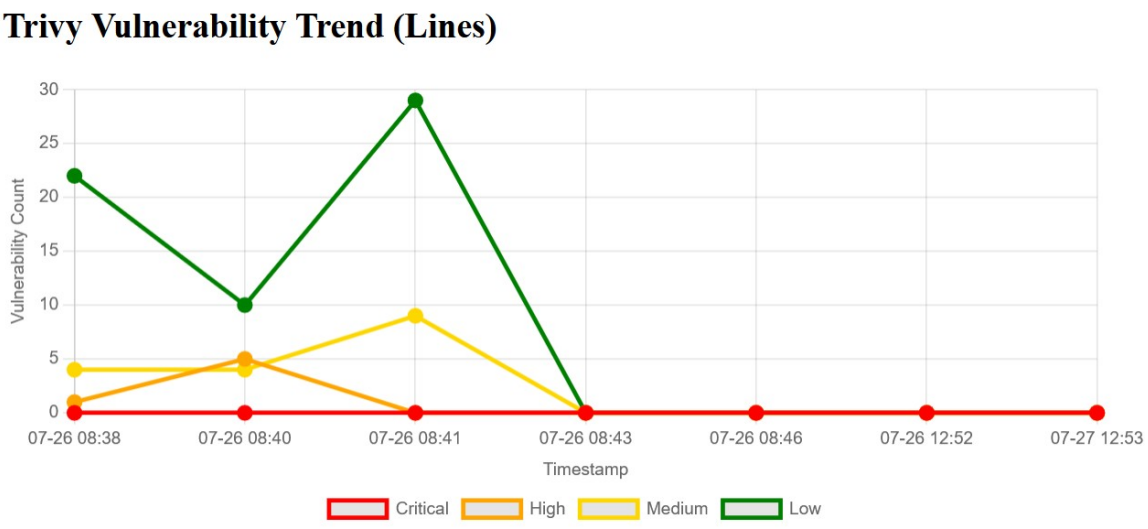

Figure 4: Trivy Vulnerability Trend (Lines)

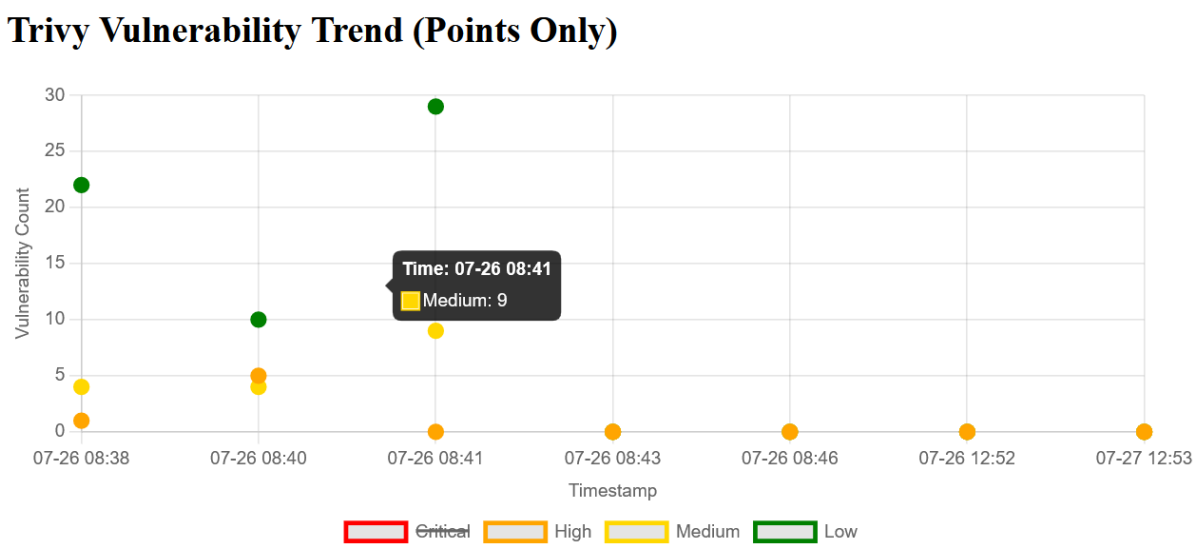

Figure 5: Trivy Vulnerability Trend (Points Only)

5. Discussion

The results demonstrate that even with minimal infrastructure, it is possible to gain significant insight into the evolving security posture of containerized applications. Our approach offers several practical advantages and surfaces important considerations regarding real-world DevSecOps adoption.

5.1. Practical Benefits

- Lightweight and accessible: The system uses only GitHub Actions, Trivy, and static CSV logs. It avoids the overhead of database backends, paid dashboards, or proprietary integrations.
- Visual awareness: By introducing time-series analysis, security becomes less reactive and more proactive. Teams can immediately see if a new dependency causes a spike in risk or if hardening measures produce measurable improvements.
- Zero infrastructure requirements: The entire solution is self-contained within a GitHub repository. No external hosting, containers, or cloud resources are needed.

5.2. Reproducibility Transparency

The full pipeline, data history, and rendered dashboard are stored in the same version-controlled project. This enables:

- Auditable security regression history (e.g., for compliance or security reviews).
- Collaboration between developers and security engineers.
- Educational use in teaching secure CI/CD practices [30].

Unlike commercial tools that hide scanning logic and history behind dashboards, our implementation allows full control and customization by the user.

5.3. Limitations

- Metric granularity: The current approach aggregates vulnerability counts by severity, but does not distinguish between fixed vs. unfixed CVEs, affected package names, or risk contexts.

- No direct correlation to code changes: Vulnerability increases may stem from base image changes or dependency drift unrelated to intentional development decisions.
- Limited historic window: By default, only the latest seven scans are stored to reduce CSV complexity. Larger history sizes may require adaptation of storage and visualization strategies.

5.4. Opportunities for Extension

Several enhancements could strengthen this solution:

- Storing full Trivy JSON per scan, enabling deep drill-down and historical forensics.
- SBOM integration, aligning scan results with specific software bills of materials.
- Correlation with Git commits or PRs, so spikes or improvements can be traced to specific changes.
- GitHub badges, e.g., to display current vulnerability counts in README.
- Multi-project support, allowing monitoring of several services from one dashboard.

This approach provides a compelling entry point for teams aiming to adopt DevSecOps incrementally, enabling observability over time without disrupting existing workflows or requiring extensive investment.

Conclusions

This paper presents a practical, reproducible method for visualizing vulnerability trends in containerized applications within CI/CD pipelines. By integrating Trivy scans with GitHub Actions and storing results in a time-series CSV format, we enable lightweight security regression trackin —a critical but often missing component of modern DevSecOps practices.

Unlike most point-in-time vulnerability detection tools, our implementation focuses on temporal awareness: understanding how a container's risk profile evolves. This empowers development teams to make informed decisions, prioritize remediation, and identify the security impact of changes between builds.

The method is especially suitable for small teams, open-source projects, educational environments, and early-stage DevSecOps adoption. It lowers the barrier to historical security monitoring without requiring cloud-native platforms or commercial licenses.

In future work, we aim to extend the system to support full scan traceability, software bill of materials (SBOM) analysis, commit-to-scan correlation, and GitHub-based alerting mechanisms. Our goal is to make historical security awareness as integral to CI/CD as build status and test coverage are today.

Declaration on Generative AI

While preparing this work, the authors used the AI programs Grammarly Pro to correct text grammar and Strike Plagiarism to search for possible plagiarism. After using this tool, the authors reviewed and edited the content as needed and took full responsibility for the publication's content.

References

[1] S. Shevchenko, et al., Information Security Risk Management using Cognitive Modeling, in: Cybersecurity Providing in Information and Telecommunication Systems II, CPITS-II, vol. 3550 (2023) 297–305.

[2] S. Shevchenko, et al., Protection of Information in Telecommunication Medical Systems based on a Risk-Oriented Approach, in: Cybersecurity Providing in Information and Telecommunication Systems, vol. 3421 (2023) 158–167.

[3] Y. Kostiuk, et al., A System for Assessing the Interdependencies of Information System Agents in Information Security Risk Management using Cognitive Maps, in: 3rd Int. Conf. on Cyber Hygiene & Conflict Management in Global Information Networks (CH&CMiGIN), Kyiv, Ukraine, vol. 3925, 2025, 249–264.
[4] Y. Kostiuk, et al., Models and Algorithms for Analyzing Information Risks during the Security Audit of Personal Data Information System, in: 3rd Int. Conf. on Cyber Hygiene & Conflict Management in Global Information Networks (CH&CMiGIN), Kyiv, Ukraine, vol. 3925, 2025, 155–171.
[5] Z. Pan et al., Ambush from All Sides: Understanding Security Threats in Open-Source Software CI/CD Pipelines, arXiv, 2024. doi:10.48550/arXiv.2401.17606
[6] A. Zerouali, T. Mens, G. Robles, and J. M. Gonzalez-Barahona, On the Relation Between Outdated Docker Containers, Severity Vulnerabilities, and Bugs, Information and Software Technology, 153, 2023.
[7] AWS, What is DevSecOps? aws.amazon.com/what-is/devsecops/
[8] Aqua Security, Trivy: Simple and Comprehensive Vulnerability Scanner. https://github.com/aquasecurity/trivy
[9] Anchore, Grype: Vulnerability Scanner for Container Images and Filesystems. https://github.com/anchore/grype
[10] C. Rajapakse, M. A. Babar, and H. Zhang, Practitioners' Perspectives on Integrating Security into DevOps, arXiv, 2021. doi:10.48550/arXiv.2107.02096
[11] GitHub Docs, Security Overview of GitHub Actions. https://docs.github.com/en/actions/security-guides/security-hardening-for-github-actions.
[12] Productplan, Technical Debt. www.productplan.com/glossary/technical-debt/
[13] GitHub Actions, About GitHub Actions. https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions
[14] M. Feio, M. Leitão, J. Fonseca, S. Guerreiro, An Empirical Study of DevSecOps Focused on Continuous Security Testing, in: Proceedings of EuroS&PW, 2024.
[15] Docker Docs, Dockerfile Best Practices. https://docs.docker.com/develop/develop-images/dockerfile_best-practices/
[16] A. Kerman, et al., Implementing a Zero-Trust Architecture, National Institute of Standards and Technology (NIST), 2020.
[17] H. Wist, K. Helsem, and D. Gligoroski, An Empirical Study on the Security of Official and Community Docker Hub Images, arXiv, 2020. doi:10.48550/arXiv.2006.02932
[18] M. N. Haque, M. A. Babar, Well Begun is Half Done: Empirical Evidence on Security Hygiene Practices in Using Base Images for Dockerfiles, arXiv, 2021. doi:10.48550/arXiv.2112.12597
[19] T. Kaur, S. Dey, V. Poenaru, J. Prins, To Use or Not to Use: An Empirical Study of Vulnerabilities in Scientific Container Images, arXiv, 2020. doi:10.48550/arXiv.2010.13970
[20] D. Shevchuk, O. Harasymchuk, A. Partyka, N. Korshun, Designing Secured Services for Authentication, Authorization, and Accounting of Users, in: Cybersecurity Providing in Information and Telecommunication Systems, 3550, 2023, 217–225.
[21] V. Lakhno, V. Kozlovskii, Y. Boiko, A. Mishchenko, I. Opirskyy, Management of Information Protection based on the Integrated Implementation of Decision Support Systems, Eastern-Eur. J. Enterp. Technol. Inf. Controlling Syst. 5(9(89)) (2017) 36–41. doi:10.15587/1729-4061.2017.111081
[22] O. Vakhula, I. Opirskyy, O. Mykhaylova, Research on Security Challenges in Cloud Environments and Solutions based on the "Security-as-Code" Approach, in: Cybersecurity Providing in Information and Telecommunication Systems, 3550, 2023, 55–69.
[23] O. Harasymchuk, O. Deineka, A. Partyka, V. Kozachok, Information Classification Framework According to SOC 2 Type II, in: Cybersecurity Providing in Information and Telecommunication Systems II, 3826, 2024, 182–189.
[24] O. Deineka, O. Harasymchuk, A. Partyka, A. Obshta, Application of LLM for Assessing the Effectiveness and Potential Risks of the Information Classification System According to SOC 2

Type II, in: Cybersecurity Providing in Information and Telecommunication Systems, 3991, 2025, 215–232.
[25] Mozilla, Working with JSON. https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Scripting/JSON
[26] Microsoft, Create or Edit .csv Files to Import into Outlook. https://support.microsoft.com/en-us/office/create-or-edit-csv-files-to-import-into-outlook-4518d70d-8fe9-46ad-94fa-1494247193c7
[27] Chartjs. Chartjs. https://www.chartjs.org/
[28] D. Fernández González, J. R. García, J. M. de Fuentes, SecDocker: Hardening the Continuous Integration Workflow, SN Comput. Sci. 3:80 , 2022. doi:10.1007/s42979-021-00939-4
[29] Amazon, Best Practices for CI/CD Pipelines. https://docs.aws.amazon.com/prescriptive-guidance/latest/strategy-cicd-litmus/cicd-best-practices.html
[30] S. Milevskyi, et al., Development of the Sociocyberphysical Systems Multi-Contour Security Methodology, Eastern-Eur. J. Enterp. Technol. 1/9 (127) (2024) 34–51.